**Question 1: What is WPF?**

**Answer:** WPF stands for Windows Presentation Foundation. It's a re-invention of a UI for Desktop applications using WPF. Apart from dropping controls on "**Windows Forms**" just as developers have been doing for years, WPF provides an extra rapid boost to the application development including Rich User Interface, Animation and much more.

In a nutshell the following things can be done using WPF:

* Draw normal controls and graphics.
* Can easily load/play audio and video files.
* Can provide smooth graphical effects such as drop shadows and color gradients.
* Can use shared styles which can be used across the same controls to provide the same theme, skin and design.
* Transforming objects including shapes, controls and video.
* Can create and animate 3D graphics.
* Can easily draw vector graphics that scale without jagged aliasing.

**Advantages**

* Tight multimedia integration
* Resolution independence
* Hardware acceleration

## **Question 2: What is Content Alignment in WPF?**

**Answer:** The content of content controls in **WPF** is dealt using various properties. These two properties are **HorizontalContentAlignment** and **VerticalContentAlignment**. These properties are defined in the **System.Windows.Controls.Control** class that is the parent class of all controls in WPF.

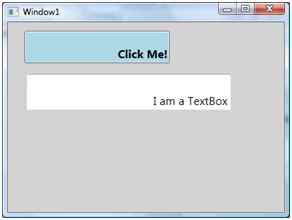
If we create a UI with a Button and a TextBox control, the UI looks like the following figure where the default vertical and horizontal alignment of content of a Button is center. The default vertical and horizontal alignment of content of a TextBox is left and top.



 You want to set the contents of the Button and TextBox to bottom and right.The code sets **VerticalContentAlignment** and **HorizontalContentAlignmentproperties** to bottom and right.

1. <Grid Name="StackPanel1" Background="LightGray">
2. <Button Name="Button1" Background="LightBlue" Height="45" Content="Click Me!" Margin="23,12,159,0" VerticalAlignment="Top" FontSize="16" FontWeight="Bold" VerticalContentAlignment="Bottom" HorizontalContentAlignment="Right" />
3. <TextBox Height="50" Margin="26,72,74,0" Name="textBox1" VerticalAlignment="Top" Text="I am a TextBox" FontSize="16" VerticalContentAlignment="Bottom" HorizontalContentAlignment="Right" />
4. </Grid>

Output that looks as in Figure:



## **Question 3: What are Resources in WPF?**

**Answer:** Windows Presentation Foundation (WPF) resources provide a simple way to reuse commonly defined objects and values. Resources in WPF allow you to set the properties of multiple controls at a time. For example, you can set the background property on several elements in a WPF application using a single resource.

The best way of defining the resources is on a Window or Page element level. Any resource that you define for an element also applies to their child elements of that element. For example, if you define a resource for a Window element that has a Grid as a child element, then the resources defined for the window elements can also be used by the grid element. However, if you define a resource for the grid element, then the resource applies only to the child elements of the grid element.

Syntax for resources in WPF,

<elementName propertyName="{markupExtension keyName}">

   <!-Content -->

</elementName>

Where,

* **elementName:** Name of the element that uses the resource.
* **propertyName:** Name of the property that takes its value from the resource.
* **markupExtension:** Define type of resource.
* **keyName:** key name of the resource, which is unique string to identify the resource.

There are two types of resource, namely,

* Static Resource
* Dynamic Resource

## **Question 4: What are static and dynamic resources?**

 There are two types of resource, namely,

* Static Resource
* Dynamic Resource

**Static Resource** : We should use the StaticResource markup extension to define the resource as a static resource. The value of StaticResource is determined at the time of loading. Let's have a sample program, Add the below code snippet in Window1.xaml file inside the Grid.

1. <Grid.Resources>
2. <SolidColorBrush x:Key="lblbgcolor" Color="Blue"/>
3. </Grid.Resources>
4. <Label Name="lbl" Margin="71,44,77,0" Background="{StaticResourcelblbgcolor}" Height="49" />

Above code, Grid control uses the Resources property (<Grid.Resources>) to define resource. SolidColorBrush resource named lblbgcolor defined. lblbgcolor resource is used to set the background property of lable.

**Dynamic Resource**

Dynamic Resource we use in a situation where we want to change the value of property at run time.Let's have a sample program, Add the following code snippet in Window1.xaml file inside the Window element.

1. <Window.Resources>
2. <SolidColorBrush x:Key="brush" Color="Red" />
3. </Window.Resources>
4. <Button x:Name="btn" Content="Click Me" Click="Button\_Click" Background="{DynamicResource brush}" Height="100" Width="100" />

Open code behind and add the following code snippet.

1. private void Button\_Click(object sender, RoutedEventArgs e)
2. {
3. this.btn.SetResourceReference(BackgroundProperty, "brush");
4. }

In the above code, Window control uses the Resources property (<Window.Resources>) to define resource. SolidColorBrush resource named brush defined. Brush resource is used to set the background property of button.

## **Question 5: What is value convertor in WPF?**

**Answer:** A Value Converter functions as a bridge between a target and a source and it is necessary when a target is bound with one source, for instance you have a text box and a button control. You want to enable or disable the button control when the text of the text box is filled or null.

 In this case you need to convert the string data to Boolean. This is possible using a Value Converter. To implement Value Converters, there is the requirement to inherit from **I Value Converter** in the **System.Windows.Data** namespace and implement the two methods **Convert** and **Convert Back.**

**Example**

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. using System.Windows.Data;
7. namespace ValueConverters
8. {
9. public class ValueConverter: IValueConverter
10. {
11. public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
12. {
13. bool isenable = true;
14. if (string.IsNullOrEmpty(value.ToString()))
15. {
16. isenable = false;
17. }
18. return isenable;
19. }
20. public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)
21. {
22. throw new NotImplementedException();
23. }
24. }
25. }

## **Question 6: What is MVVM?**

Answer: MVVM (Model View ViewModel) is a framework for making applications in WPF. MVVM is the same as the MVC framework. It is a 3-tier architecture plus one more layer. We can do loose coupling using MVVM.

MVVM was introduced by John Gossman in 2005 specifically for use with WPF as a concrete application of Martin Fowler's broader Presentation Model pattern. The implementation of an application, based on the MVVM patterns, uses various platform capabilities that are available in some form for WPF, Silverlight Desktop/web, and on Windows. Many commercial applications, including Microsoft Expression products, were built following MVVM.

**Advantage of MVVM**

* Modularity
* Test driven approach.
* Separation UI and Business layer as view and view model.
* Code sharing between pages and forms.
* Easy to Maintain.

**List of features of MVVM**

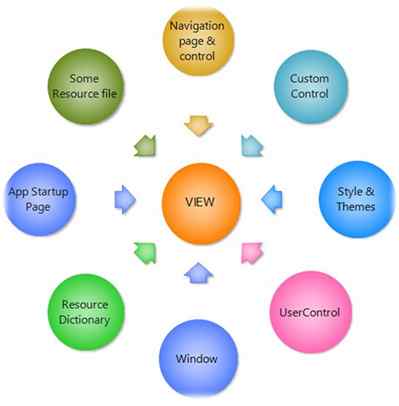
* It separates the business and presentation layers, like MVP and MVC.
* Improve Structure/separation of concerns (View, ViewModel and Model).
* Enable a better Design/Developer Workflow.
* Enhance simplicity and testability.
* Enabled by the robust data binding capability of XAML.
* No need to use a code behind file (minimalist code-behind file).
* Provides application development ability for multiple environments.
* Powerful Data Binding, command, validation and much more.
* The designer and developer can work together.

## **Question 7: How can you explain view and view model in MVVM?**

**Answer:** The View is the client interface, input-output interface or the user interface. It collects all the user interface elements of the window, navigation page, user control, resource file, style and themes, custom tools and controls. The view is unaware of the ViewModel and the Model, and vice versa the ViewModel and Model is unaware of the View and control is tightly decoupled.But the view model is aware of the needs of the view. They communicate by data binding and a dependency property or properties.

**Question 8: ViewModel in MVVM?**

**Answer** : ViewModel is a non-visual class. The MVVM Design Pattern does not derive from any WPF or Silverlight based class. The ViewModel is unaware of the view directly. Communication between the View and ViewModel is through some property and binding. Models are connected directly to the ViewModel and invoke a method by the model class, it knows what the model has, like properties, methods etcetera and also is aware of what the view needs.One View-Model can connect to multiple models, work like a one-to-many relationship and encapsulate business logic and data for the View. A ViewModel inherits some interface like **INotifyPropertyChanged**, **icommand INotifyCollectionChanged** etcetera.



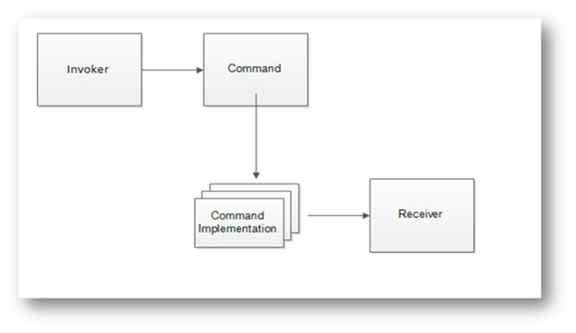
## **Question 8: What is Command Design Pattern and ICommand in WPF?**

**Answer: C**ommand pattern is one of the most powerful design patterns in object oriented design patterns. This pattern allows you to decouple the request of an action from the object that actually performs an action, in other words a command pattern represents an action as an object. A Command object does not contain the functionality that is to be executed. This removes the direct link between the command definitions and the functionality and it’s promoting a loose coupling. When you want to implement operations based on the user request then the command pattern is the best pattern to handle your object.

 The following are the members of the Command Design Pattern:

* Client
* Invoker
* Command
* Concrete Command
* Receiver

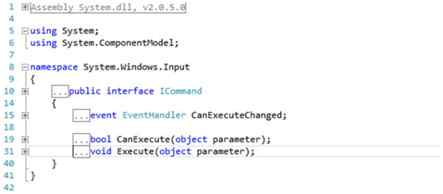
**Flow of Command Design Pattern**



**ICommand**

An ICommand is a core component of MVVM. ICommand is frequently used in MVVM, it provides separation logic between a View and View Model (User Interface and Business Logic). XAML offers a way to bind your GUI event better by ICommand. ICommand requires the user to define two methods, bool CanExecute and void Execute. The CanExecute method really just says to the user, can I execute this Action? This is useful for controlling the context in which you can perform GUI actions.

ICommand is very simple but it’s more interesting and complicated when you are using it in an application. ICommand integrates your User Interface to business logic or it’s making a direct communication between a View to a View Model. It also provides a mechanism for the view to update the model/view-model.



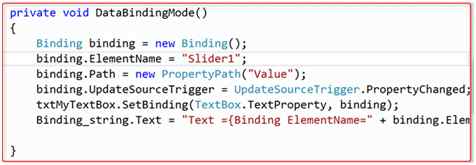
## **Question 9: What is the Data Binding concept and How Binding works in WPF?**

**Answer:** Data Binding is one of the greatest and most powerful features of XAML (WPF, Silverlight, Windows Phone or Windows 8) compared to other traditional web and Windows app technology in .NET. There was simple data binding for displaying single values, and complex data binding for displaying and formatting a bunch of data. XAML (WPF, Silverlight, Windows Phone or Windows 8 ) however makes it easy to bind nearly any property to any element or object or data source. You take data from some property or object or dependency property and bind it to another dependency property or object or else directly to an element. In a single word you can say in XAML "Data binding is the process of getting information from one object to another and displaying it in one or more elements in the user interface".

**How {Binding} works in WPF**

The Binding keyword looks as in the following image. Here is the binding of TextBox UI controls with some binding property source object.

source object Binding to a WPF element from code behind.

 In the above sample a TextBox and slider from code behind using some binding property.

**Some Very Useful Properties of the Binding Class**

|  |  |
| --- | --- |
| Property Name | Description |
| Element Name | The name of the element that gets or sets the binding source object when binding to a XAML element. |
| FallbackValue | Set the value to use when the binding does not return values. |
| Converter | Set the converter for the UI element. |
| Mode | Set the Binding diection between the target and source objects. |
| Path | Get or Set the path to the source property of the Binding source. |
| RelativeSource | Gets or Sets the binding source by specifying it's location and relative to the postion of the binding target. |
| Source | Gets or Sets the binding source when not binding to a WPF element. |
| StringFormat |  |
| UpdateSourceTrigger |  |
| ValidationRules |  |
| NotifyOnSourceUpdated | Gets or sets the value determining the direction of the dataflow in the binding. |
| NotifyOnTargetUpdated | Gets or sets a value that indicates whether to raise the source Update event when a value is transferred from the source to the target. |

## **Question 10: What is Trigger and how many types of triggers in WPF?**

**Answer:** A Trigger is typically used in a Style or Control Template. It triggers on properties of whatever is being templated, and sets other properties of the control (or of specific template elements). For example, you would use a Trigger on **IsMouseOver** to respond to the mouse being over the control, and the setters might update a brush to show a "**hot**" effect.

**Why to use Trigger**

Triggers are used in styles to perform actions on a change of any property value or event fires. Triggers create visual effects on controls. By using Triggers we can change the appearance of Framework Elements.There are **five** types of triggers supported by WPF; they are:

1. Property Trigger
2. Data Trigger
3. MultiTrigger
4. MultiDataTrigger
5. Event Trigger

**Example 1:** For example, let's say you have a rectangle control. You want to change the background color of that control when the mouse hovers over the rectangle control and revert back when mouse leaves. For this you need to code in the mouse hover event and mouse leave event of the rectangle control in the backend class for changing the color of rectangle as in the following code.

1. private void Rectangle\_MouseMove\_1(object sender, MouseEventArgs e)
2. {
3. this.rctback.Fill = Brushes.Red;
4. }
5. private void Rectangle\_MouseLeave(object sender, MouseEventArgs e)
6. {
7. this.rctback.Fill = Brushes.Green;
8. }

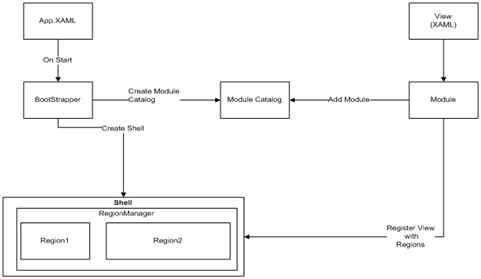
In the preceding code you have filled the background color of the Rectangle control by writing code in two different events, but a trigger helps to overcome this problem by reducing the code.

**Question 11: What is Prism in WPF?**

**Answer:** Prism (Composite Application Guidance for WPF and Silverlight) is designed to build applications in WPF and Silverlight that have a single code base. It helps to develop the client application in a modular fashion so that complexity of a large application can be divided into simpler modules.

In other words “Prism is developed by Microsoft Patterns and Practices and provides guidance designed to help you to more easily design and build rich, flexible and easy-to-maintain Windows Presentation Foundation (WPF) desktop applications.”.

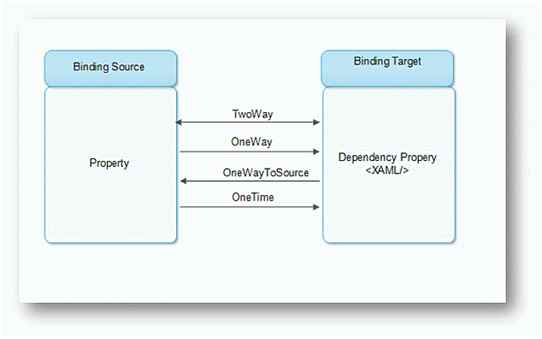
**Architecture:** The following diagram shows basic architecture:



1. **App.XAML:** Call Boot Strapper on Application\_Startup.
2. **BootStrapper:** This is a class file that calls Shell (Shell.XAML) and so creates catalogue of module.
3. **Shell:** This is like a Master Page having regions.
4. **Region:** It is like placeholders to register views.
5. **View:** This is XAML file having User Interface
6. **Module:** Each module can have one or more View(s) which are registered to Region (in the Shell) through Region Manager.

**Question 12: What are the Binding Modes in XAML?**

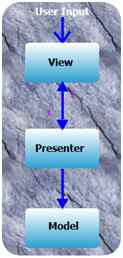
**Answer:** The DataBinding mode defines the communication direction to the source or the direction of data flow from the source. In XAML (WPF, Silverlight, WP or Win8 App) there are five ways you can bind a data target object to a source.



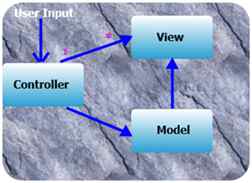
* **OneWay:** Data moves only one direction, the source property automatically updates the target property but the source is not changed.
* **TwoWay:** Data moves both directions, if you change it in the source or target it is automatically updated to the other.
* **OneWayToSource:** Data moves from the target to the source changes to the target property to automatically update the source property but the target is not changed.
* **OneTime:** Data is changed only one time and after that it is never set again, only the first time changes to the source property automatically update the target property but the source is not changed and subsequent changes do not affect the target property.

## **Question 13: What is the difference between MVP, MVC and MVVM?**

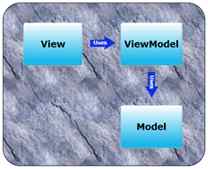
**MVP (Model-View-Presenter)** : In the MVP pattern the User sends the input to the view, the view forward it to presenter and presenter then modify the view or the model depending on the type of user action. The view and the presenter are tightly coupled through bi-directional relationship. The model does not know about the presenter. The view itself is passive, thats why it's called presenter pattern, since the presenter pushes the data into the view.



**MVC (Model-View-Controller)** :In this pattern there is only one controller that gets all the inputs directly, it modifies the data in the model depending upon the type of the input. Both the model and the view are created by the controller. The view only knows about the model, but the model does not know about any other objects.



The Model View ViewModel (MVVM) is an architectural pattern used in software engineering that originated from Microsoft which is specialized in the Presentation Model design pattern. It is based on the Model-view-controller pattern (MVC), MVVM is a way of creating client applications that leverages core features of the WPF platform, allows for simple unit testing of application functionality, and helps developers and designers work together with less technical difficulties.



## **Question 14: What are the Templates in WPF ?**

**Answer:** Templates are an integral part of user interface design in WPF. WPF has the following three types of templates:

* Control Template
* Items Panel Template
* Data Template

**Control Template**

 The ControlTemplate of a control defines the appearance of the control. We can change or define a new look and appearance of a control by simply changing the ControlTemplate of a control. ControlTemplates are even more useful when you write your own controls. Using ControlTemplates, you can build a custom button that has a circular layout and changes its color when you mouse over or press it. The ControlTemplate element in XAML defines a ControlTemplate at design-time. Templates are usually defined as resources using a FrameworkElement's Resources property. The following code snippet is the syntax for defining a ControlTemplate for a Button element.

1. <Grid>
2. <Grid.Resources>
3. <ControlTemplate x:Key="RoundButtonTemplate" />
4. </Grid.Resources>
5. </Grid>

We need to create a circular button where the outer circle of the button is of a different color than the inner circle and when you mouse over and press the button, it changes the background color.



Add a Grid as contents of the ControlTemplate. Add two Ellipse elements within a Grid with different radii and different color fills.

1. <Grid.Resources>
2. <ControlTemplate x:Key="RoundButtonTemplate">
3. <Grid>
4. <Ellipse Width="100" Height="100" Name="ButtonBorder" Fill="OrangeRed" />
5. <Ellipse Width="80" Height="80" Fill="Orange" /> </Grid>
6. </ControlTemplate>
7. </Grid.Resources>

The following code snippet creates a Button element and sets its Template to the ControlTemplate that we created- <Button Template="{StaticResource RoundButtonTemplate}" >OK</Button>

**ItemsPanelTemplate** : In the previous example, we saw how a Style element can be used within the resources to group multiple properties of elements and set them using the Style property of elements. However, Style functionality does not end here. Style can be used to group and share not only properties, but also resources and event handlers on any FrameworkElement or FrameworkContentElement. Styles are resources and used as any other resource and can be applied to the current element, parent element, root element and even on the application level. The scope if styles are similar to any other resources. The resource lookup process first looks up for local styles and if not found, it traverses to the parent element in the logical tree and so on. In the end, the resource lookup process looks for styles in the application and themes. The Style element in XAML represents a style. The typical definition of the Style element looks as in the following:

<Style>

   Setters

</Style>

As you can see from the definition of Style, a Style has one more Setter element. Each Setter consists of a property and a value. The property is the name of the property and the value is the actual value of that property of the element to that the style will be applied to.

**Setters Property** : The Setters property of Type represents a collection of Setter and EventSetter objects. Listing 4 uses the Setters property and adds a Setter and EventSetter object.

The code snippet in Listing 4 sets the Setters property of a Style by adding a few Setter elements and one EventSetter element using XAML at design-time.

1. <Grid>
2. <Grid.Resources>
3. <Style TargetType="{x:Type Button}">
4. <Setter Property="Width" Value="200"/> <Setter Property="Height" Value="30"/> <Setter Property="Foreground" Value="White"/> <Setter Property="Background" Value="DarkGreen"/> <Setter Property="BorderBrush" Value="Black"/> <EventSetter Event="Click" Handler="Button1\_Click"/>
5. </Style>
6. </Grid.Resources>
7. <Button>Click me</Button>
8. </Grid>

## **Question 15: What are the various layout panels in WPF ?**

**Answer:** WPF comes with the following five built-in panels:

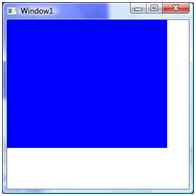
* Canvas
* DockPanel
* Grid
* StackPanel
* WrapPanel

The purpose and use of these panels is different. Each panel has a different way to position and reposition child controls placed within that panel. The following articles in this series will summarise these panels and their usages.Similar to any other WPF control, a Panel control may be represented in two ways. First, at design-time using XAML elements and attributes, and second, at run-time, using a WPF class and its properties.

 The code snippet in Listing 2 creates a Grid panel at design-time using XAML.

1. <Window x:Class="CanvasPanelSample.Window1"
2. xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3. xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4. Title="Window1" Height="300" Width="300"
5. Name="RootWindow">
6. <Grid Name="GridPanel" Background="Blue"
7. Width="250" Height="200"
8. VerticalAlignment="Top"
9. HorizontalAlignment="Left"
10. FlowDirection="LeftToRight"
11. />
12. </Window>

**Listing 2** generates **Figure 2**.



**Question 16: What is Attached Properties in WPF ?**

**Answer:** Attached properties are basically Dependency Properties that allows the attachment of a value to any random object. Attached Properties (AP) are again a kind of Dependency Property (DP) in XAML. They can be used to receive a notification of a change of themself since they are a type of Dependency Property but one of the differences that these properties have is that they are not defined in the same class they used, unlike DPs.

* The type that defines the Attached Property is designed so that it can be the parent element of the elements that will set values for the Attached Property. The type then iterates its child objects using internal logic against some object's tree structure, obtains the values and acts on those values in some manner.
* The type that defines the Attached Property will be used as the child element for a variety of possible parent elements and content models.
* The type that defines the Attached Property represents a service. Other types set values for the Attached Property. Then, when the element that set the property is evaluated in the context of the service, the Attached Property values are obtained using internal logic of the service class.

APs are called “**attached**” properties because we can attach some behaviour to the control that is originally not expected from that control.

**Example** : Suppose I have a TextBox control and I want to extend its functionality to accept letters and special symbols but not numeric values.

The AP that I have defined in my class **TextBlockExtension.cs** is as in the following:

1. public static bool GetAllowOnlyString(DependencyObject obj)
2. {
3. return (bool) obj.GetValue(AllowOnlyStringProperty);
4. }
5. public static void SetAllowOnlyString(DependencyObject obj, bool value)
6. {
7. obj.SetValue(AllowOnlyStringProperty, value);
8. }
9. // Using a DependencyProperty as the backing store for AllowOnlyString. This enables animation, styling, binding, etc...
10. public static readonly DependencyProperty AllowOnlyStringProperty = DependencyProperty.RegisterAttached("AllowOnlyString", typeof (bool), typeof (TextblockExtension), new PropertyMetadata(false, AllowOnlyString));
11. private static void AllowOnlyString(DependencyObject d, DependencyPropertyChangedEventArgs e)
12. {
13. if (d is TextBox)
14. {
15. TextBox txtObj = (TextBox) d;
16. txtObj.TextChanged += (s, arg) =>
17. {
18. TextBox txt = s as TextBox;
19. if (!Regex.IsMatch(txt.Text, "^[a-zA-Z]\*$"))
20. {
21. txtObj.BorderBrush = Brushes.Red;
22. MessageBox.Show("Only letter allowed!");
23. }
24. };
25. }
26. }

In the code, as we can see, the AP has the default value of False, in other words we need to provide the APs the value true wherever I want this functionality to work for the TextBox.

In my **MainWindow.xaml.cs** I have defined my TextBox as in the following:

1. <TextBox Width="200" Height="50" local:TextblockExtension.AllowOnlyString="True"></TextBox>

## **Question 17: What is resource in WPF? How many types of resources in WPF?**

**Answer:** Windows Presentation Foundation (WPF) resources provide a simple way to reuse commonly defined objects and values. Resources in WPF allow you to set the properties of multiple controls at a time. For example, you can set the background property on several elements in a WPF application using a single resource.The best way of defining the resources is on a Window or Page element level. Any resource that you define for an element also applies to their child elements of that element.If you define a resource for the grid element, then the resource applies only to the child elements of the grid element.

Syntax for resources in WPF is as follows:

<elementName propertyName="{markupExtension keyName}">

<!-Content -->

</elementName>

There are two types of resource, namely,

* Static Resource
* Dynamic Resource

**Static Resource**

 We should use the StaticResource markup extension to define the resource as a static resource. The value of StaticResource is determined at the time of loading.

1. <Grid.Resources>
2. <SolidColorBrush x:Key="lblbgcolor" Color="Blue"/>
3. </Grid.Resources>
4. <Label Name="lbl" Margin="71,44,77,0" Background="{StaticResourcelblbgcolor}" Height="49" />

**Dynamic Resource** : Dynamic Resource we use in a situation where we want to change the value of property at run time.

1. <Window.Resources>
2. <SolidColorBrush x:Key="brush" Color="Red" />
3. </Window.Resources>
4. <Button x:Name="btn" Content="Click Me" Click="Button\_Click" Background="{DynamicResource brush}" Height="100" Width="100" />

**Open Code behind and add the following code snippet:**

1. private void Button\_Click(object sender, RoutedEventArgs e)
2. {
3. this.btn.SetResourceReference(BackgroundProperty, "brush");
4. }

## **Question 18: What is the difference between Static and Dynamic resources?**

**Answer** : The most basic difference is that StaticResource evaluates the resource one time only, but DynamicResource evaluates it every time the resource is required. And due to this reason, DyanamicResource is heavy on the system but it makes pages or windows load faster.

**Static Resources** : A Static Resource will be resolved and assigned to the property during the loading of the XAML that occurs before the application is actually run. It will only be assigned once and any changes to the resource dictionary is ignored.Static resource references work best for the following circumstances:

* Your application design concentrates most of its resources into page or application level resource dictionaries. Static resource references are not re-evaluated based on runtime behaviours such as reloading a page, and therefore there can be some performance benefit to avoiding large numbers of dynamic resource references when they are not necessary per your resource and application design.
* You are setting the value of a property that is not on a Dependency Object or a freezable.
* You are creating a resource dictionary that will be compiled into a DLL, and packaged as part of the application or shared between applications.

**Dynamic Resources**

A Dynamic Resource assigns an Expression object to the property during loading but does not actually lookup the resource until runtime when the Expression object is asked for the value. This defers looking up the resource until it is needed at runtime.

Dynamic resources work best for the following circumstances:

* The value of the resource depends on conditions that are not known until runtime. This includes system resources, or resources that are otherwise user settable. For example, you can create setter values that refer to system properties, as exposed by System Colours, System Fonts, or System Parameters. These values are truly dynamic because they ultimately come from the runtime environment of the user and operating system. You might also have application-level themes that can change, where page-level resource access must also capture the change.
* You are creating or referencing theme styles for a custom control.
* You intend to adjust the contents of a Resource Dictionary during an application lifetime.

**Example**

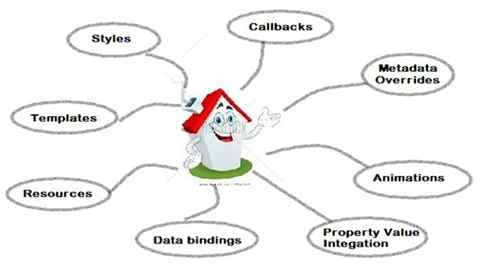
1. <Window.Resources>
2. <SolidColorBrush x:Key="brush" Color="Green" />
3. <Style TargetType="Border" x:Key="PageBackground">
4. <Setter Property="Background" Value="Gold"/>
5. </Style>
6. </Window.Resources>
7. <Grid>
8. <Border Style="{DynamicResource PageBackground}">
9. <Button x:Name="btn" Content="Rajkumar Test" Click="Button\_Click" Background="{DynamicResource brush}" Height="30" Margin="53,130,85,130" /> </Border>
10. </Grid>

You can apply on code behind like the following,

1. private void Button\_Click(object sender, RoutedEventArgs e)
2. {
3. this.btn.SetResourceReference(BackgroundProperty, "brush");
4. }

## **Question 19: What is WPF Dependency Property and how can we use?**

**Answer:** WPF has provided some extended services to the CLR property that we can collectively call Dependency Properties. A Dependency Property is a property whose value depends on the external sources, such as animation, data binding, styles, or visual tree inheritance. Not only this, but a Dependency Property also has the builtin feature of providing notification when the property has changed, data binding and styling.



**Advantages of a Dependency Property:**

* Less memory consumption
* Property value inheritance
* Change notification and Data Bindings
* Participation in animation, styles and templates
* CallBacks
* Resources
* Overriding Metadata

**Code**

1. public class CarDependencyProperty: DependencyObject
2. {
3. //Register Dependency Property
4. public static readonly DependencyProperty CarDependency = DependencyProperty.Register("MyProperty", typeof (string), typeof (DependencyPropertySample));
5. public string MyCar
6. {
7. get
8. {
9. return (string) GetValue(CarDependency);
10. }
11. set
12. {
13. SetValue(CarDependency, value);
14. }
15. }
16. }
17. public partial class CarDependencyPropertyDemo: Window
18. {
19. public CarDependencyPropertyDemo()
20. {
21. InitializeComponent();
22. }
23. private void MyButton\_Click(object sender, RoutedEventArgs e)
24. {
25. CarDependency dpSample = TryFindResource("CarDependency ") as CarDependency;
26. MessageBox.Show(dpSample.MyCar);
27. }
28. }

## **Question 20: What is Attached Properties and how to register it?**

**Answer:** Attached Properties (AP) can be used to receive a notification of a change of themself since they are a type of Dependency Property but one of the differences that these properties have is that they are not defined in the same class they used, unlike DPs. One of the misconceptions that a WPF developer usually has is that these properties can only be defined in the parent control the control in which we want to use them.

AP can be defined in one of the following three contexts:

* The type that defines the Attached Property is designed so that it can be the parent element of the elements that will set values for the Attached Property. The type then iterates its child objects using internal logic against some object's tree structure, obtains the values and acts on those values in some manner.
* The type that defines the Attached Property will be used as the child element for a variety of possible parent elements and content models.
* The type that defines the Attached Property represents a service. Other types set values for the Attached Property. Then, when the element that set the property is evaluated in the context of the service, the Attached Property values are obtained using internal logic of the service class.

**Example**

1. public static bool GetAllowOnlyString(DependencyObject obj)
2. {
3. return (bool) obj.GetValue(AllowOnlyStringProperty);
4. }
5. public static void SetAllowOnlyString(DependencyObject obj, bool value)
6. {
7. obj.SetValue(AllowOnlyStringProperty, value);
8. }
9. // Using a DependencyProperty as the backing store for AllowOnlyString. This enables animation, styling, binding, etc...
10. public static readonly DependencyProperty AllowOnlyStringProperty = DependencyProperty.RegisterAttached("AllowOnlyString", typeof (bool), typeof (TextblockExtension), new PropertyMetadata(false, AllowOnlyString));
11. private static void AllowOnlyString(DependencyObject d, DependencyPropertyChangedEventArgs e)
12. {
13. if (d is TextBox)
14. {
15. TextBox txtObj = (TextBox) d;
16. txtObj.TextChanged += (s, arg) =>
17. {
18. TextBox txt = s as TextBox;
19. if (!Regex.IsMatch(txt.Text, "^[a-zA-Z]\*$"))
20. {
21. txtObj.BorderBrush = Brushes.Red;
22. MessageBox.Show("Only letter allowed!");
23. }
24. };
25. }
26. }

In the code, as we can see, the AP has the default value of False, in other words we need to provide the APs the value true wherever I want this functionality to work for the TextBox.

In my MainWindow.xaml.cs I have defined my TextBox as in the following:

1. <TextBox Width="200" Height="50" local:TextblockExtension.AllowOnlyString="True"></TextBox>

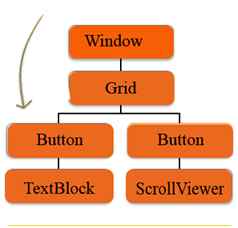
**Question 21: What is a Routed event?**

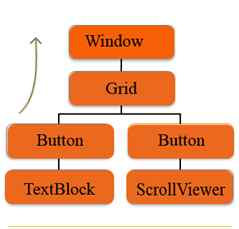
Answer: Routed Events is about the Hierarchy of the controls you are using in the Events. Routed Events are a new Infrastructure given by WPF that permit events to tunnel down the tree to the target elements or Bubble up to the Root element. Routed Events are just like normal events.

**Types of Routed Events** :Routed Events are of three types, which are as follows:

* Bubbling Events
* Tunneling Events
* Direct Events

**Tunneling Events:** Tunneling events are the reverse of the Bubbling events. Tunneling Events raised first in the controls hierarchy. These events are raised by the Root elements. This allows events to tunnel down the tree.

 **Bubbling Events:** Bubbling Events are those Events which are first raised by the control than raised by the other controls in the control hierarchy. It allows Bubble up to the tree till the Root Element. First Tunneling events are raised then bubbling events raised.



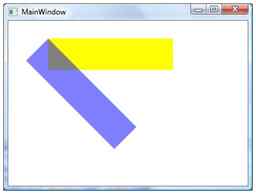
**Direct Event:** Direct Event is generally raised by the control itself. The behavior of this event is same as the .NET general event.

**Question 22: What is Rotate transform in WPF?**

**Answer:** RotateTransform rotates an element clockwise by a specified angle about the point. The RotateTransform object in WPF represents RotateTransform. The Angle property represents the angle in degrees to rotate clockwise. The CenterX and CenterY properties represent the X and Y coordinates of the center point. By default, a ScaleTransform is centered at the point (0,0), which corresponds to the upper-left corner of the rectangle.

 Creates two rectangles with same position and sizes accept the second rectangle is rotated at 45 degrees.

1. <Grid>
2. <!-- Original Rectangle -->
3. <Rectangle Width="200" Height="50" Fill="Yellow" />
4. <!-- Rectangle with 45 degrees rotation -->
5. <Rectangle Width="200" Height="50" Fill="Blue" Opacity="0.5">
6. <Rectangle.RenderTransform>
7. <RotateTransform CenterX="0" CenterY="0" Angle="45" /> </Rectangle.RenderTransform>
8. </Rectangle>
9. </Grid>



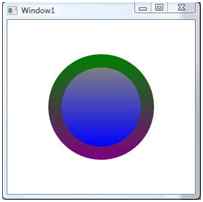
 The following code snippet changes the values of CenterX and CenterY.

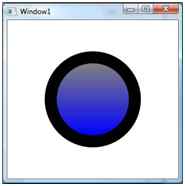
1. <Rectangle Width="200" Height="50" Fill="Blue" Opacity="0.5" Margin="61,27,117,184">
2. <Rectangle.RenderTransform>
3. <RotateTransform CenterX="-50" CenterY="50" Angle="45" />
4. </Rectangle.RenderTransform>
5. </Rectangle>

## **Question 23: What is the Control Template in WPF?**

**Answer:** The ControlTemplate contains the tree of elements that define the desired look. After you define a ControlTemplate you can attach it to any Control or Page by setting it's TemplateProperty.

1. <Grid>
2. <Grid.Resources>
3. <ControlTemplate x:Key="buttonTemplate">
4. <Grid>
5. <Ellipse Width="160" Height="160" x:Name="outerCircle">
6. <Ellipse.Fill>
7. <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
8. <GradientStop Offset="0" Color="Green"></GradientStop>
9. <GradientStop Offset="1" Color="Purple"></GradientStop>
10. </LinearGradientBrush>
11. </Ellipse.Fill>
12. </Ellipse>
13. <Ellipse Width="120" Height="120">
14. <Ellipse.Fill>
15. <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
16. <GradientStop Offset="0" Color="Gray"> </GradientStop>
17. <GradientStop Offset="1" Color="Blue"> </GradientStop>
18. </LinearGradientBrush>
19. </Ellipse.Fill>
20. </Ellipse>
21. </Grid>
22. <ControlTemplate.Triggers>
23. <Trigger Property="Button.IsMouseOver" Value="True">
24. <Setter TargetName="outerCircle" Property="Fill" Value="Black"> </Setter>
25. </Trigger>
26. <Trigger Property="Button.IsPressed" Value="True">
27. <Setter Property="RenderTransform">
28. <Setter.Value>
29. <ScaleTransform ScaleX=".8" ScaleY=".8"> </ScaleTransform>
30. </Setter.Value>
31. </Setter>
32. <Setter Property="RenderTransformOrigin" Value=".6,.6"> </Setter>
33. </Trigger>
34. </ControlTemplate.Triggers>
35. </ControlTemplate>
36. </Grid.Resources>
37. <Button Template="{StaticResource buttonTemplate}">Click Me</Button>
38. </Grid>

 **After MouseOver:**

****

**Question 24: How can we create Borderless Window in WPF?**

**Answer:** We can create a borderless window in two ways.

Firstly, by writing a WindowStyle property = None, SingleBorderWindow, ThreeDBorderWindow or ToolWindow in the <Window> element.

1. <Window x:Class="WpfApp1.MainWindow" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="MainWindow" Height="350" Width="525" WindowStyle="None">
2. <Grid> </Grid>
3. </Window>

The second way is to open the Property Window, select the window style property to either None, SingleBorderWindow, ThreeDBorderWindow or ToolWindow.



**Question 25: What is XAML in WPF and also explain the types of XAML?**

**Answer:** Extensible Application Markup Language and pronounced "zammel" is a markup language used to instantiate .NET objects. Although XAML is a technology that can be applied to many different problem domains, its primary role in life is to construct WPF user interfaces.

Important task XAML performs as as follows,

* **Wiring up an event handler:** Attaching event handler in the most cases, for example Click on Button is easy to do in Visual Studio. However once we understand how events are wired up in XAML,we'll be able create more sophisticated connections.
* **Defining resources:** Resources are the object which once we define in XAML can be re-used in the various places inside markup. Resources allow us to centralize and standardize formatting, and create nonvisual objects such as templates and animations.
* **Defining control template:** WPF controls are designed to be lookless, which means we can substitute our custom visuals in place of the standard appearance. To do so, we must create our own control template, which is nothing more than a block of XAML markup.

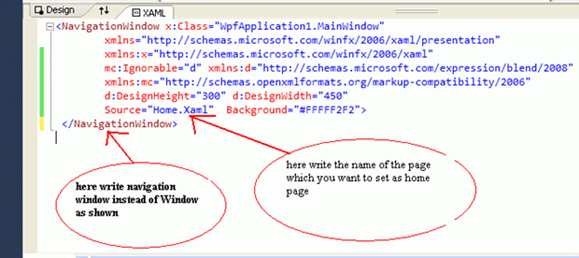
**Type of XAML**

* **WPF XAML:** Encompasses the elements that describe WPF content, such as vector graphics, controls, and documents.
* **XPS XAML:** It is the part of WPF XAML that defines an XML representation for formatted electronic documents.
* **Silverlight XAML:** A subset of WPF XAML that's intended for Silverlight applications. Silverlight is a cross-platform browser plug-in that allows us to create rich web content.
* **WF XAML:** Encompasses the elements that describe Windows Workflow Foundation.

## **Question 26: How to set content of a NavigationWindow?**

**Answer:**The Navigation Window class is derived from the Window class, so it inherits all the properties of Windows such as methods, properties and events. The navigation window provides backward and forward buttons for navigating to pages that we have visited before or have yet to visit.

**Creating Navigation Paged Application:** To create Navigation Window based applications, use a Navigation Window container instead of a Window container as shown in the following picture and the source is the property of the Navigation window and write the name of the page that you want to set as the home page as shown in the following picture:



## **Question 27: What is a WPF Child Window?**

Answer: A ChildWindow control is a light weight window that can be used as a child window or a popup control. The parent window is automatically disabled when a child window is active and modal. You can think of a ChildWindow as a custom modal or modeless dialog where you can place any child controls you want. However, the ChildWindow has several common Window properties.

**Creating a ChildWindow:** The ChildWindow element represents a WPF ChildWindow control in XAML. The ChildWindow control is defined in the System.Windows.Controls namespace.

creates a simple ChildWindow control with its Width, Height, Name, IsModal and Caption property.

1. <wpfx:ChildWindow Name="PopupChildWindow" Caption="Child Window" Width="300" Height="200"IsModal="True" />

We need to call the Show() method to make a Child Window visible.

1. PopupChildWindow.Show();

**Question 28: What are the WPF Content Controls?**

**Answer:** Content Controls are mainly parent containers to hold the content. It displays information to the user to be viewed but that generally won't be modified.

With content controls, the following aspects:

1. **Property:** A Property specifies the object's appearance or behavior. For example, the IsEnabled property specifies whether the user can interact with a control or not.
2. **Method:** A Method is a routine that a control executes to perform something. For example Coa unt Method counts the number of items available for the object or control.
3. **Event:** An event is related when a control raises one to let your application know that something has happened. For example TextBox raises a TextChanged event whenever its text changes.

**Code Snippet**

1. <Button Name="btnAdd" Content="Add" Click="btnAdd\_Click" />

See for more details:

## **Question 29: What is the Tab Control in WPF?**

**Answer:** The Tab control is a common UI element that has been around for some time. It makes a convenient way to organize your window when there is more than could realistically fit and still be comprehensible. Tab Control is easier in Windows Presentation Foundation.

Two elements play main roles in building a tab control:

* TabControl and
* TabItem

TabControl is the container of one or more TabItem elements like as follows.

1. <TabControl>
2. <TabItem Header="Tab 1">xyz</TabItem>
3. <TabItem Header="Tab 2">abc</TabItem>
4. </TabControl>

In WPF, Tabs are very easy to implement. Create a new WPF Window, remove the default Grid tags, and add the following XAML:

1. <TabControl>
2. <TabItem Header="Tab 1">xyz</TabItem>
3. <TabItem Header="Tab 2">abc</TabItem>
4. </TabControl>

## **Question 30: How can I clip or crop an image?**

Answer: Clipping a region is a process of displaying partial area of a region by setting the outline of a region. In WPF, the clipping has been extended to all elements that are inherited from UIElement that includes controls, images, panels, Windows, and Pages.

1. Window.Clip>
2. <EllipseGeometry Center="150,160" RadiusX="120" RadiusY="120" />
3. </Window.Clip>

The following code snippet clips or crops an image to an ellipse.

1. <Image Source="Garden.jpg">
2. <Image.Clip>
3. <EllipseGeometry Center="150,160" RadiusX="120" RadiusY="120" />
4. </Image.Clip>
5. </Image>
6. private void ClipImage()
7. {
8. // Create a BitmapImage
9. BitmapImage bmpImage = new BitmapImage();
10. bmpImage.BeginInit();
11. bmpImage.UriSource = new Uri(@ "C:\Images\Garden.jpg", UriKind.RelativeOrAbsolute);
12. bmpImage.EndInit();
13. // Clipped Image
14. Image clippedImage = new Image();
15. clippedImage.Source = bmpImage;
16. EllipseGeometry clipGeometry = new EllipseGeometry(new Point(150, 160), 120, 120);
17. clippedImage.Clip = clipGeometry;
18. LayoutRoot.Children.Add(clippedImage);
19. }

## **Question 31: What are Converters in WPF?**

**Answer:** Converters provide substantial supremacy since they allow insertion of an object between a source and a target object. At a high level, converters are a chunk of custom code hooked up using the binding and the data will flow via that converter. So, whenever data is flown from a source to a target, one can change the value or can change the type of object that needs to be set on the target property.

So, whenever data travels from source to target, it can be transformed in two ways:

1. **Data value:** Here the transformation will be done with just the value by keeping the data type intact. For example, for number fields, you can transform a value from a floating point number to an integer by keeping the actual value as a float.
2. **Data type:** One can also transform the data type. For example, setting a style based on some Boolean flag.

**Defining a converter** : Defining any converter requires implementation of an IValueConverter interface in a class. This interface has the following two methods:

1. **Convert:** Is called when data is flowing from a source to a target.
2. **ConvertBack:** Is called when data is flowing from a target to a source. It is basically useful in two-way binding scenarios.

**How to use**

 Once your class is part of a ResourceDictionary, you can point to the instance of the converter using the Converter property of Binding, along with a StaticResource markup extension.

**Where to place converters** : To implement an IValueConverter one must create a class and then put the instance of that class in a ResourceDictionary within your UI.

## **Question 32: How does “UpdateSourceTrigger” affect bindings?**

**Answer:** This is a property on a binding that controls the data flow from a target to a source and used for two-way data binding. The default mode is when the focus changes but there are many other options available.

Properties available with UpdateSourceTrigger

* **Default:** This is the default value and it means a lost focus for most of the controls.
* **LostFocus:** Value update will be on hold until the focus moves out of the control.
* **PropertyChanged:** Value update will happen whenever a target property changes. It usually happen on every keystoke.
* **Explicit:** Used to defer source updates until the user does it forcibly by the click of a button or so.

**Default vs LostFocus**

Default and LostFocus means the same thing for most of the controls with the exception of DataGrid. For DataGrid:

* **Lost Focus**: Cell lost focus
* **Default:** Row lost focus

**Code**

1. <grid>
2. <Grid.ColumnDefinitions>
3. <ColumnDefinition Width="50\*" />
4. <ColumnDefinition Width="50\*" />
5. </Grid.ColumnDefinitions>
6. <TextBlock Text="Source:" Width="auto" />
7. <TextBox Name="SourceText" Width="160" Height="30" Margin="48,0,44,82" />
8. <TextBlock Text="Target:" Grid.Column="1" Width="auto" />
9. <TextBox Name="TargetText" Width="160" Height="30" Text="{Binding ElementName=SourceText, Path=Text,UpdateSourceTrigger=Default}" Grid.Column="1" Margin="44,0,47,82" />
10. </grid>

**Output** : When the user types into the source TextBox:

**Question 33: What are various ways of doing alignment in WPF?**

**Answer:** The FrameworkElement has two alignment properties: HorizontalAlignment and Vertical Alignment. The Horizontal Alignment property is a type of HorizontalAlignment enumeration and represents how a child element is positioned within a parent element horizontally.The HorizontalAlignment enumeration has the four properties Left, Center, Right and Stretch. The Left, Center and Right properties sets a child element to left,center and right of the parent element. The Stretch property stretches a child element to fill the parent element's allocated layout space.**Example**

1. <StackPanel Background="LightGray">
2. <Button Name="Rect1" Background="LightBlue" Width="150" Height="50" HorizontalAlignment="Left" Content="Left Aligned" />
3. <Button Name="Rect2" Background="LightGreen" Width="150" Height="50" HorizontalAlignment="Center" Content="Center Aligned" />
4. <Button Name="Rect3" Background="LightCyan" Width="150" Height="50" HorizontalAlignment="Right" Content="Right Aligned" />
5. <Button Name="Rect4" Background="LightPink" Height="50" HorizontalAlignment="Stretch" Content="Stretch Aligned" />
6. </StackPanel>

The **VerticalAlignment** property is a type of **HorizontalAlignment** enumeration and represents how a child element is positioned within a parent element vertically.

 The **VerticalAlignment** enumeration has the four properties Top, Center, Bottom and Stretch. The Top, Center and Bottom properties set a child element to top, center or bottom of the parent element. The Stretch property stretches a child element to fill the parent element's allocated layout space vertically.**Example**

1. <Grid>
2. <Grid.ColumnDefinitions>
3. <ColumnDefinition Width="100" />
4. <ColumnDefinition Width="100" />
5. <ColumnDefinition Width="100" />
6. <ColumnDefinition Width="100" />
7. </Grid.ColumnDefinitions>
8. <Button Name="Button1" Background="LightBlue" Height="30" Width="100" VerticalAlignment="Top" Content="Left Aligned" />
9. <Button Name="Button2" Background="LightGreen" Height="30" Width="100" Grid.Column="1" VerticalAlignment="Center" Content="Center Aligned" />
10. <Button Name="Button3" Background="LightCyan" VerticalAlignment="Bottom" Height="30" Width="100" Grid.Column="2" HorizontalAlignment="Left" Content="Right Aligned" />
11. <Button Name="Button4" Background="LightPink" Content="Stretch ligned" Width="100" Grid.Column="3" HorizontalAlignment="Stretch" />
12. </Grid>

## **Question 34: What is ToolTip and how we use it in WPF?**

**Answer:** A ToolTip control shows some information or a hint about a control in a floating box when the mouse hovers over that control and it disappears when the mouse is moved away from that control. Tool tips are used to help the users to learn the purpose of Controls in many Windows-based programs. Tooltips are nothing but small rectangles that appear when the user hovers the mouse pointer over the Control. The rectangle contains a short piece of text describing the control. Once displayed, the tips disappear when the user moves the mouse pointer away from the linked control or clicks a mouse button, or after a short delay.

**ToolTip Example:** Drag a Button control from the toolbox and drop it. Add a ToolTip property to the button with a message you want to show on mouse hover of the button. We can add a ToolTip in two ways. First the button shows the simplest way to display a ToolTip. The second method is preferred for creating a customized ToolTip.

1. <Button Content="Click Here" Margin="30" FontSize="16" ToolTip="Click Here"></Button>
2. <Button Content="Click Here" Margin="30" FontSize="16">
3. <Button.ToolTip>
4. <ToolTip> Click Here </ToolTip>
5. </Button.ToolTip>
6. </Button>

**Question 35: How can ListBox be made to scroll smoothly?**

**Answer:** A ListBox control is an items control that works as a ListBox control but only one item from the collection is visible at a time and clicking on the ListBox makes the collection visible and allows users to pick an item from the collection. Unlike a ListBox control, a ListBox does not have multiple item selection.

The ListBox element represents a ListBox control in XAML.

<ListBox></ListBox>

The Width and Height properties represent the width and the height of a ListBox. The x:Name property represents the name of the control, which is a unique identifier of a control. The Margin property sets the location of a ListBox on the parent control. The HorizontalAlignment and VerticalAlignment properties are used to set horizontal and vertical alignments.

 The code sets the vertical and horizontal alignment of the ListBox and sets the margin.

1. <ListBox x:Name="ListBox1" Width="200" Height="200"
2. VerticalAlignment="Top" HorizontalAlignment="Left"
3. Margin="10,10,0,0">
4. </ListBox>

## **Question 36: What is a Popup window and how to open and close the popup window?**

**Answer:** A popup window is a window that floats over a page or window providing functionality for some quick action. For example, a login control on a page or window or an animated popup tip of a control.The Popup element of XAML represents a WPF Popup control.

<Popup></Popup>

The Width and the Height properties represent the width and the height of a Popup. The Name property represents the name of the control that is a unique identifier of a control. The Margin property is used to set the location of a Popup on the parent control. The HorizontalAlignment and VerticalAlignment properties are used to set horizontal and vertical alignments.

The following code snippet sets the name, height, and width of a Popup control. The code also sets horizontal alignment to left and vertical alignment to top.

1. <Popup Margin="10,10,0,13" Name="Popup1" HorizontalAlignment="Left"
2. VerticalAlignment="Top" Width="194" Height="200" />

**Open a Popup:** The Popup Control displays its contents when IsOpen set true.

1. privatevoid Popup\_Ok\_Click(object sender, RoutedEventArgs e)
2. {
3. Popup1.IsOpen = true;
4. }

Where popup1 is the name of the popup control.

**Closing a Popup:** The Popup Control displays its contents when IsOpen is set to false.

1. privatevoid Popup\_Ok\_Click(object sender, RoutedEventArgs e)
2. {
3. Popup1.IsOpen = false;
4. }

## **Question 37: What is a Decorators class in WCF?**

**Answer:** Decorator class that is a simple base class of WPF layout controls. Decorators have a single child control to which they apply a single child control.

Add a click event to the button. Replace the Button's XAML with the following:

1. <Border Name="MyBorder" BorderBrush="Black" BorderThickness="4" CornerRadius="10">
2. <Button Click="Button\_Click">Hello, world!</Button>
3. </Border>

When the program is executed the window that appears similar to the image below:

**Question 38: What are the advantages and disadvantages of WPF?**

**Advantages**

* **Tight multimedia integration:** To use 3-D graphics, video, speech, and rich document viewing in Windows 32 or Windows Forms applications, you would need to learn several independent technologies and blend them together without much built-in support. WPF applications allow you to use all these features with a consistent programming model.
* **Resolution independence:** WPF lets you shrink or enlarge elements on the screen, independent of the screen's resolution. It uses vector graphics to make your applications resolution-independent.
* **Hardware acceleration:** WPF is built on top of Direct3D, which offloads work to graphics processing units (GPUs) instead of central processor units (CPUs). This provides WPF applications with the benefit of hardware acceleration, permitting smoother graphics and enhanced performance.
* **Declarative programming:** WPF uses Extensible Application Markup Language (XAML) declarative programming to define the layout of application objects and to represent 3-D models, among other things. This allows graphic designers to directly contribute to the look and feel of WPF applications.
* **Rich composition and customization:** WPF controls are easily customizable. You need not write any code to customize controls in very unique ways. WPF also lets you create skins for applications that have radically different looks.
* **Easy deployment:** WPF provides options for deploying traditional Windows applications (using Windows Installer or Click Once). This feature is not unique to WPF, but is still an important component of the technology.
* **Culturally aware controls:** Static text in controls and the return data for the String function are modified according to the culture and language specified by the end user's operating system.

**Disadvantages**

* WPF's in-box control suite is far more limited than that of WinForms.
* There's greater support in the 3rd-party control space for WinForms. (That's changing, but for now by advantage of time, WinForms has greater support in the community).
* Most developers already know WinForms; WPF provides a new learning curve.
* WPF will not run on Windows 2000 or lower.
* No MDI child mode.

## **Question 39: What are the WPF assemblies and Namespace?**

|  |  |
| --- | --- |
| WPF Assembly | Meaning in Life |
| WindowsBase.dll | Defines the base infrastructure of WPF, including dependency properties support. While this assembly contains types used within the WPF framework, the majority of these types can be used within other .NET applications. |
| PresentationCore.dll | This assembly defines numerous types that constitute the foundation of the WPF GUI layer. |
| PresentationFoundation.dll | This assembly—the ‘meatiest’ of the three—defines the WPF controls types, animation and multimedia support, data binding support, and other WPF services. For all practical purposes, this is the assembly you will spend most of your time working with directly. |

Although these assemblies provide hundreds of types within numerous namespaces, consider this partial list of WPF namespaces:

* You will encounter other namespaces during the remainder of this class.
* Again, consult the .NET Framework SDK documentation for full details.

|  |  |
| --- | --- |
| WPF Namespace | Meaning in Life |
| System.Windows | Here you will find core types such as Application and Window that are required by any WPF desktop project. |
| System.Windows.Controls | Here you will find all of the expected WPF widgets, including types to build menu systems, tool tips, and numerous layout managers. |
| System.Windows.Markup | This namespace defines a number of types that allow XAML markup and the equivalent binary format, BAML, to be parsed. |
| System.Windows.Media | Within these namespaces you will find types to work with animations, 3D rendering, text rendering, and other multimedia primitives. |
| System.Windows.Navigation | This namespace provides types to account for the navigation logic employed by XAML browser applications / desktop navigation apps. |
| System.Windows.Shapes | This namespace defines basic geometric shapes (Rectangle, Polygon, etc.) used by various aspects of the WPF framework. |

## **Question 40: What is the difference between WPF and Silverlight?**

**Answer:** Silverlight and Windows Presentation Foundation (WPF) are 2 different products from Microsoft, but have lot of overlap. Silverlight is a subset of WPF in terms of features and functionality. Silverlight was of course known as WPF/E where E means everywhere. Both use XAML, a form of XML to define controls but WPF is purely for windows while Silverlight runs in the browser on Windows and Macs. So, here are the differences between WPF and Silverlight:

1. Silverlight is meant to be used online, while WPF is for local use.
2. Silverlight lacks access to local resources, while WPF can utilize local resources.
3. Silverlight only has perspective 3D support, while WPF is capable of full 3D images.
4. Silverlight does not support some of the more advanced concepts of WPF such as controls and templating.
5. Silverlight integrates right into an HTML page whereas WPF XAML files have to be loaded via a frame if they want to mix with HTML content.

## **Question 41: What is WPF accesstext Control?**

**Answer:** The AccessText control in WPF converts a character preceded by an underscore to an Access Key. The Access Key is registered and therefore raises an event when pressed.

**Creating an AccessText**

 The AccessText element represents an AccessText control in XAML.

<AccessText>\_Click Me</AccessText>

The following code snippet adds an AccessText to a Button control. The button control click event will be raised when you select the ALT+C keys on the keyboard.

1. <Button Name="Button1" Width="120" Height="50" Margin="33,70,59,124" FontSize="16" Click="Button1\_Click">
2. <AccessText>\_Click Me</AccessText>
3. </Button>

If there are multiple underscore characters, only the first one is converted into an AccessKey; the other underscores appear as normal text. If the underscore that you want converted to the access key is not the first underscore, use two consecutive underscores for any underscores that precede the one that you want to convert.

**Question 42: What are the different types of brushes that WPF offers?**

**Answer:** Brushes and pens are objects used to draw and fill graphics objects. WPF and XAML work together and there is an WPF class corresponding to each XAML control. For example, <SolidColorBrush> tag in XAML and SolidColorBrush class in WPF, both represent the solid color brush. You can create and use brushes in both ways seperately or mix them.

In XAML and WPF model provides the following brush objects:

1. SolidColorBrush
2. LinearGradientBrush
3. RadialGradientBrush
4. DrawingBrush
5. Visual Brush
6. ImageBrush

## **Question 43: How many types of Bitmap Effects in WPF?**

**Answer:** Bitmap effects are simple pixel processing operations. A bitmap effect takes a BitmapSource as an input and produces a new BitmapSource after applying the effect.Bitmap effects enable designers and developers to apply visual effects to rendered Microsoft Windows Presentation Foundation (WPF) content. For example, bitmap effects allow you to easily apply a DropShadowBitmapEffect effect or a blur effect to an image or a button. The unmanaged APIs provide an extensible framework for independent hardware vendors (IHVs) to develop custom effects to use in WPF applications.

The following are the available Bitmap Effects in WPF:

1. BlurBitmapEffect
2. OuterGlowBitmapEffect
3. DropShadowBitmapEffect
4. BevelBitmapEffect
5. EmbossBitmapEffect

Each bitmap effect has properties that can control the filtering properties, such as Radius of BlurBitmapEffect.

**Question 44: What is tree view in WPF?How we can delete a tree view in WPF?**

**Answer:** A TreeView represents data in a hierarchical view in a parent child relationship where a parent node can be expanded or collapse. The left side bar of Windows Explorer is an example of a TreeView.

The TreeView tag represents a WPF TreeView control in XAML.

<TreeView></TreeView>

**Adding TreeView Items**

 A TreeView control hosts a collection of TreeViewItem. The Header property is the text of the item that is displayed on the view. The following code snippet adds a parent item and six child items to a TreeView control.

1. <TreeView Margin="10,10,0,13" Name="TreeView1" HorizontalAlignment="Left" VerticalAlignment="Top" Width="194" Height="200">
2. <TreeViewItem Header="Cold Drinks">
3. <TreeViewItem Header="Coke"></TreeViewItem>
4. <TreeViewItem Header="Pepsi"></TreeViewItem>
5. <TreeViewItem Header="Orange Juice"></TreeViewItem>
6. <TreeViewItem Header="Milk"></TreeViewItem>
7. <TreeViewItem Header="Iced Tea"></TreeViewItem>
8. <TreeViewItem Header="Mango Shake"></TreeViewItem>
9. </TreeViewItem>
10. </TreeView>

**Deleting TreeView Items**

We can use TreeView.Items.Remove or TreeView.Items.RemoveAt method to delete an item from the collection of items in the TreeView. The RemoveAt method takes the index of the item in the collection.

1. <Button Height="23" Margin="226,14,124,0" Name="DeleteButton"
2. VerticalAlignment="Top" Click="DeleteButton\_Click">
3. Delete Item</Button>
4. private void DeleteButton\_Click(object sender, RoutedEventArgs e)
5. {
6. TreeView1.Items.RemoveAt
7. (TreeView1.Items.IndexOf(TreeView1.SelectedItem));
8. }

The above code removes root items from the TreeView, not the subitems. To remove sub items, first we need to find the selected item and then we need to call TreeViewItem.Items.RemoveAt method.

**Question 45: What are the Commands in WPF?**

**Answer:** Commands have several purposes. The first purpose is to separate the semantics and the object that invokes a command from the logic that executes the command. This allows for multiple and disparate sources to invoke the same command logic, and it allows the command logic to be customized for different targets. For example, the editing operations Copy, Cut, and Paste, which are found in many applications, can be invoked by using different user actions if they are implemented by using commands. An application might allow a user to cut selected objects or text by either clicking a button, choosing an item in a menu, or using a key combination, such as CTRL+X. By using commands, you can bind each type of user action to the same logic.

Commands are used to share grouped actions within an application in different ways. Sometimes we need to perform the same activity, WPF provides us a feature called Command to make our work easier and faster.

There are basically four type of Commands:

* Application Commands
* Edit Commands
* Component Commands
* Media Commands

**XAML**

1. <StackPanel>
2. <Menu>
3. <MenuItem Command="ApplicationCommands.Paste" /> </Menu>
4. <TextBox />
5. </StackPanel>

**C#**

1. / Creating the UI objects
2. StackPanel mainStackPanel = new StackPanel();
3. TextBox pasteTextBox = new TextBox();
4. Menu stackPanelMenu = new Menu();
5. MenuItem pasteMenuItem = new MenuItem();
7. // Adding objects to the panel and the menu
8. stackPanelMenu.Items.Add(pasteMenuItem);
9. mainStackPanel.Children.Add(stackPanelMenu);
10. mainStackPanel.Children.Add(pasteTextBox);
12. // Setting the command to the Paste command
13. pasteMenuItem.Command = ApplicationCommands.Paste;
15. // Setting the command target to the TextBox
16. pasteMenuItem.CommandTarget = pasteTextBox;

## **Question 46: What are the properties of canvas panel in WPF?**

**Answer:** The Canvas is the most basic layout panel in WPF. It's child elements are positioned by **explicit coordinates**. The coordinates can be specified **relative to any side** of the panel usind the Canvas.Left, Canvas.Top, Canvas.Bottom andCanvas.Right attached properties.

 The panel is typically used to group 2D graphic elements together and not to layout user interface elements. This is important because specifing absolute coordinates brings you in trouble when you begin to resize, scale or localize your application. People coming from WinForms are familiar with this kind of layout - but it's not a good practice in WPF.

In WPF, a Canvas Panel has a very simple layout. We can position any object using its two properties:

* Canvas.Left
* Canvas.Top

**Note:** A Canvas panel does not resize automatically when our normal application resizes in the browser.

 Here is an example of a Canvas panel:

1. <Canvas Background="Pink" Width="250" Height="100">
2. </Canvas>

**Question47: What is WPF TextBlock?**

**Answer:** The TextBlock control is one of the most fundamental controls in WPF, yet it's very useful. It allows you to put text on the screen, much like a Label control does, but in a simpler and less resource demanding way. A common understanding is that a Label is for short, one-line texts (but may include e.g. an image), while the TextBlock works very well for multiline strings as well, but can only contain text (strings). Both the Label and the TextBlock offers their own unique advantages, so what you should use very much depends on the situation.

**Creating a TextBlock**

The TextBlock element represents a WPF TextBlock control in XAML.

 <TextBlock/>

The Width and Height attributes of the TextBlock element represent the width and the height of a TextBlock. The Text property of the TextBlock element represents the content of a TextBlock. The Name attribute represents the name of the control, which is a unique identifier of a control. The Foreground property sets the foreground color of contents. This control does not have a Background property.

1. <TextBlock Name="TextBlock1" Height="30" Width="200"
2. Text="Hello! I am a TextBlock." Foreground="Red">
3. </TextBlock>

The output looks like the following,

**Question 48: What is XAML TabControl in WPF?**

**Answer:** A Tab Control has tab items and each tab item represents a container that is used to host other controls.The WPF TabControl allows you to split your interface up into different areas, each accessible by clicking on the tab header, usually positioned at the top of the control. Tab controls are commonly used in Windows applications and even within Windows' own interfaces, like the properties dialog for files/folders etc.

Just like with most other WPF controls, the TabControl is very easy to get started with. Here's a very basic example:

1. <Window x:Class="WpfTutorialSamples.Misc\_controls.TabControlSample" <Window x:Class="WpfTutorialSamples.Misc\_controls.TabControlSample" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="TabControlSample" Height="200" Width="250">
2. <Grid>
3. <TabControl>
4. <TabItem Header="General">
5. <Label Content="Content goes here..." /> </TabItem>
6. <TabItem Header="Security" />
7. <TabItem Header="Details" /> </TabControl>
8. </Grid>
9. </Window>

**Question 49: What is Virtualization in WPF?**

**Answer:** Virtualization technique in WPF improves the rendering performance of UI elements. By applying virtualization, the layout system ensures that only the visible items of a container are rendered on the screen. For example, a list control may have thousands of items but virtualization will reduce the rendering to the visible items only.

**VirtualizingStackPanel:** The VirtualizingStackPanel control in WPF is used to implement virtualization.The IsVirtualizing property of the VirtualizingStackPanel activates the virtualization. By default, the IsVirtualizing property is set to true. When IsVirtualizing is set to false, a VirtualizingStackPanel behaves the same as an ordinary StackPanel.

1. <VirtualizingStackPanel Width="300" Height="200" />

The VirtualizingStackPanel.VirtualizationMode property has two values, Standard and Recycling. The default value of VirtualizationMode is Standard and means that the VirtualizingStackPanel creates an item container for each visible item and discards it when it is no longer needed (such as when the item is scrolled out of view). When an ItemsControl contains many items, the process of creating and discarding item containers can degrade performance. In that case, using the Recycling reuses item containers instead of creating a new one each time.

**Question 50: What is DockPanel Control in WPF?**

**Answer:** A DockPanel is a panel where each child element docks to one of the four edges. DockPanel enables docking of child elements to an entire side of the panel streching it to fill the entire height or width.

 The DockPanel makes it easy to dock content in all four directions (top, bottom, left and right). This makes it a great choice in many situations, where you want to divide the window into specific areas, especially because by default, the last element inside the DockPanel, unless this feature is specifically disabled, will automatically fill the rest of the space (center).

1. <Window x:Class="DockPanelExample1.MainWindow" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="DockPanel Example" Height="350" Width="525">
2. <DockPanel>
3. <Button Content="Button1"></Button>
4. <Button Content="Button2"></Button>
5. <Button Content="Button3"></Button>
6. <Button Content="Button4"></Button>
7. <Button Content="Button5"></Button>
8. <Button Content="Button6"></Button>
9. </DockPanel>
10. </Window>

**Preview**

**Question 51: What is XAML StatusBar?**

**Answer:** XAML StatusBar represents a status bar. A StatusBar is a horizontal window that usually sits at the bottom of a window to display various kinds of status information of an application.

 The **StatusBar** element in **XAML** represents a **WPF StatusBar** control.

<StatusBar></StatusBar>

 The Width and Height properties represent the width and the height of a StatusBar. The Name property represents the name of the control that is a unique identifier of a control.

 The following code snippet creates a StatusBar control and set its content to some text.

1. <StatusBar Name="McSBar" Height="30" VerticalAlignment="Bottom"
2. Background="LightBlue" >
3. This is a status bar
4. </StatusBar>

The output looks as in the following figure:

**Question 52: Describe Polyline in WPF?**

**Answer:** A polyline is an object in AutoCAD that consists of one or more line (or arc) segments. A rectangle is an example of a polyline that you are already familiar with. As you've seen, it is one object that can be modified and worked with easier than four separate lines.

 In other words, A polyline is a collection of connected straight lines. The Polyline object represents a polyline shape and draws a polyline with the given points. The Points property represents the points in a polyline. The Stroke property sets the color and StrokeThickness represents the width of the line of a polyline.

**Creating a Polyline:** The Polyline element in XAML creates a polyline shape. The following code snippet creates a polyline by setting its Points property. The code also sets the black stroke of width 4.

1. <Polyline
2. Points="10,100 100,200 200,30 250,200 200,150"
3. Stroke="Black"
4. StrokeThickness="4" />

The Polyline element in XAML creates a polyline shape. The following code snippet creates a polyline by setting its Points property. The code also sets the black stroke of width 4.

1. <Polyline
2. Points="10,100 100,200 200,30 250,200 200,150"
3. Stroke="Black"
4. StrokeThickness="4" />